Configuration Manual

MSc Research Project
MSCFTD1 – Practicum Part 2

Vivek Kumar
Student ID: x23100311

School of Computing
National College of Ireland

Supervisor: Faithful Onwuegbuche

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Vivek Kumar
Student ID:	x23100311
Programme:	MSCFTD1 – Practicum Part 2 **Year:** 2023-2024
Module:	MSc Research Project
Lecturer:	Faithful Onwuegbuche
Submission Due Date:	12/08/2024
Project Title:	Predictive Modeling for Financial Distress in Indian Small Cap Stocks
Word Count:	913 **Page Count:** 11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Vivek

Date: 9/8/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	✓
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	✓
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	✓

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Vivek Kumar
Student ID: x23100311

1 Introduction

The Goal of this manual is to give a clear idea of the configuration parameters and the context in which they are applied. This manual is intended for academic research on topic "Predictive Modeling for Financial Distress in Indian Small Cap Stocks".

2 System Configuration

2.1 Hardware Requirements

To ensure optimal performance of the system, the following hardware utilized:

- Processor**:** 12th Gen Intel(R) Core(TM) i7-1260P 2.10 GHz
- Memory (RAM): 16.0 GB (15.7 GB usable)
- **Storage**: 1 TB HDD

2.2 Software Requirements

- System type: 64-bit operating system, x64-based processor
- Windows 11 Home
- Python notebook
- Google colab
- Libraries such as pandas, numpy, matplot, StandardScaler, SimpleImputer, SMOTE, Counter, PCA, MinMaxScaler

2.3 Development Environment

Jupyter Notebook in Google Colab used for interactive development and testing.

3 Project Implementation

3.1 Data Collection

Data was collected from publicly available source such as screen [1] and in few cases by reviewing financial statements from annual report of respective small cap companies.

- Format: Excel worksheet converted to CSV to process dataset in Google Colab

3.2 Data Pre-processing

Steps and techniques for cleaning and preparing data:

[1] https://www.screener.in/

- Handling Missing Values: Imputation (mean, median) based on previous 5 years of dataset the missing values where imputed for each rows.

```
# Handling Missing Values with Imputation
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
```

- Normalization and Scaling: Min-Max scaling used to normalize the dataset

```
[21] scaler = MinMaxScaler()
     X_scaled = scaler.fit_transform(X)
```

- Data Cleaning: Removing duplicates, correcting errors, and filtering outliers

```
[6] # Printing column names to inspect for any issues
    print(data.columns.tolist())

['Name', 'CMP\xa0Rs.', 'Debt / Eq', 'Pledged\xa0%', 'Int Coverage', 'ROE 5Yr Var\xa0%', 'Chg in Prom Hold 3Yr\xa0%', 'ROE 5Yr\xa0%', 'Profit Var 5Yrs\xa0%', 'EPS Var 5Yrs\

[7] # Removing non-breaking spaces from column names
    data.columns = data.columns.str.replace('\xa0', ' ', regex=False)

    # Verify column names after cleaning
    print("Cleaned column names:", data.columns.tolist())

Cleaned column names: ['Name', 'CMP Rs.', 'Debt / Eq', 'Pledged %', 'Int Coverage', 'ROE 5Yr Var %', 'Chg in Prom Hold 3Yr %', 'ROE 5Yr %', 'Profit Var 5Yrs %', 'EPS Var 5
```

3.3 Feature Selection

Methods for selecting relevant features:

- Techniques – In the dataset 14 different ratios defined as features

```
[50] # Defining features and target
     features = ['Debt / Eq', 'Pledged %', 'Int Coverage', 'ROE 5Yr Var %',
                 'Chg in Prom Hold 3Yr %', 'ROE 5Yr %', 'Profit Var 5Yrs %',
                 'EPS Var 5Yrs %', 'Free Cash Flow 5Yrs Rs.Cr.', 'CMP / BV',
                 'ROCE 5Yr %', '5Yrs PE', 'Mar Cap Rs.Cr.']

     X = data[features]
```

Also dropped name of the companies from features as these are not ratios

```
[20] # Extracting the financial ratios (excluding the 'Name' column)
     financial_ratios = data.drop(['Name'], axis=1)

     # Handling missing values by filling with the mean of each column
     financial_ratios.fillna(financial_ratios.mean(), inplace=True)
```

```
[27] # Standardizing the financial ratios
     scaler = StandardScaler()
     standardized_data = scaler.fit_transform(financial_ratios)
```

3.4 Feature Engineering

Creating and modifying features: Principal Component Analysis (PCA) used to identify which financial ratios contribute most to the variability in the data and thus may be important indicators of financial distress.

```
from sklearn.decomposition import PCA

# Applying PCA
pca = PCA(n_components=len(financial_ratios.columns))  # Number of components is equal to the number of ratios
principal_components = pca.fit_transform(standardized_data)
```

```
[17] # Explaining variance by each principal component
     explained_variance = pca.explained_variance_ratio_
     cumulative_explained_variance = explained_variance.cumsum()

     # Creating a DataFrame to display the explained variance
     explained_variance_df = pd.DataFrame({
         'Principal Component': [f'PC{i+1}' for i in range(len(explained_variance))],
         'Explained Variance': explained_variance,
         'Cumulative Explained Variance': cumulative_explained_variance
     })

     print(explained_variance_df)
```

- Results from PCA

```
                              PC1       PC2       PC3       PC4       PC5  \
CMP Rs.                  0.017020 -0.136658  0.076098 -0.002869  0.569741
Debt / Eq                0.009255  0.413238  0.506915  0.245208 -0.024679
Pledged %               -0.024414  0.163840 -0.155685  0.269152 -0.040749
Int Coverage             0.042948 -0.104881  0.014866 -0.083894 -0.441200
ROE 5Yr Var %            0.512641  0.197985 -0.206731  0.009195  0.039288
Chg in Prom Hold 3Yr %  -0.048051  0.039023 -0.003994 -0.122678 -0.134197
ROE 5Yr %                0.243716 -0.504538  0.287635  0.098925 -0.055884
Profit Var 5Yrs %        0.561887  0.044002 -0.070897  0.003714  0.026109
EPS Var 5Yrs %           0.561756  0.082863 -0.112906  0.018185  0.028755
Free Cash Flow 5Yrs Rs.Cr. -0.055589 -0.057430 -0.288174  0.581025 -0.109012
CMP / BV                 0.078196  0.316894  0.564274  0.274104 -0.020743
ROCE 5Yr %               0.163119 -0.477554  0.325195  0.113686 -0.160447
5Yrs PE                  0.003479  0.320153 -0.048922 -0.145977  0.152619
Mar Cap Rs.Cr.          -0.032360 -0.183039  0.026473  0.211689  0.620510
Distress_Label          -0.067220 -0.029235 -0.243374  0.585960 -0.088891

                              PC6       PC7       PC8       PC9      PC10  \
CMP Rs.                  0.292716  0.427690 -0.264785  0.441483  0.327649
Debt / Eq                0.038648 -0.026400 -0.110557  0.032909 -0.147254
Pledged %               -0.439179 -0.031327 -0.617571 -0.262031  0.435267
Int Coverage             0.267062  0.699995 -0.247057 -0.332742 -0.209774
ROE 5Yr Var %            0.098022 -0.026572 -0.066271 -0.008910 -0.037242
Chg in Prom Hold 3Yr %   0.724578 -0.425771 -0.111934 -0.266402  0.398513
ROE 5Yr %               -0.049128 -0.084971  0.099295 -0.063867  0.115869
Profit Var 5Yrs %       -0.023345 -0.023724  0.043071  0.005628 -0.028426
EPS Var 5Yrs %           0.043220 -0.019504 -0.035248  0.016398 -0.042619
Free Cash Flow 5Yrs Rs.Cr.  0.161975  0.116211  0.111972  0.029994  0.144728
CMP / BV                 0.102386  0.059504  0.058249 -0.000957  0.042317
ROCE 5Yr %              -0.172693  0.001133  0.102226 -0.079096  0.316123
5Yrs PE                 -0.119248  0.339410  0.590195 -0.355248  0.462285
Mar Cap Rs.Cr.           0.076913 -0.052940 -0.048418 -0.629341 -0.342627
Distress_Label           0.149994  0.014523  0.261345  0.140626 -0.086472

                             PC11      PC12      PC13      PC14      PC15
CMP Rs.                  0.082877  0.037082  0.029801  0.011369  0.019922
Debt / Eq                0.058876  0.468431  0.498249 -0.051809  0.020973
Pledged %                0.162255 -0.116031  0.022037  0.023687  0.003387
Int Coverage             0.085719 -0.010739  0.013183  0.017442  0.014958
ROE 5Yr Var %           -0.039880  0.099998 -0.175936 -0.712140  0.296269
Chg in Prom Hold 3Yr %   0.084900  0.035985  0.021710  0.074064  0.005213
ROE 5Yr %                0.097893 -0.456585  0.519687 -0.253721  0.030020
Profit Var 5Yrs %        0.034032 -0.010029  0.074862  0.628570  0.520441
EPS Var 5Yrs %          -0.002936  0.021190  0.030620  0.137320 -0.798643
Free Cash Flow 5Yrs Rs.Cr. -0.676504  0.031087  0.163644  0.036719  0.014763
CMP / BV                -0.117972 -0.484037 -0.479967  0.039819 -0.011122
ROCE 5Yr %              -0.015884  0.553293 -0.384449  0.000817 -0.016214
5Yrs PE                  0.087920  0.021240  0.148343 -0.042282 -0.029344
Mar Cap Rs.Cr.          -0.025507  0.057800 -0.075781  0.025263 -0.002314
Distress_Label           0.675452  0.011031 -0.097268 -0.000232 -0.007996
```

```
[29] # Transforming the original data into the principal component space
     principal_components_df = pd.DataFrame(data=principal_components, columns=[f'PC{i+1}' for i in range(principal_components.shape[1])])
```

```
[30] # To get the loadings (coefficients of the original variables in the principal components)
     loadings = pd.DataFrame(pca.components_.T, index=financial_ratios.columns, columns=[f'PC{i+1}' for i in range(principal_components.shape[1])])

     # Displaying the loadings
     print(loadings)
```

- Defining target variables: For distress criteria, determined if a company is distressed based on PCA scores. A company is considered distressed if at least `threshold` out of the top 5 PCA scores are negative.

```python
import numpy as np

def is_distressed(pca_scores, threshold=5):
    """
    Determine if a company is distressed based on PCA scores.
    A company is considered distressed if at least `threshold` out of the top 5 PCA scores are negative.
    """
    # Count how many of the top 5 PCA component scores are negative
    num_negative = np.sum(pca_scores[:5] < 0)
    return num_negative >= threshold

# Applying distress labeling
data['Distress_Label'] = [is_distressed(scores) for scores in X_pca]

# Converting boolean to integer (1 for distress, 0 for no distress)
data['Distress_Label'] = data['Distress_Label'].astype(int)

# Printing labeled data for verification
print(data[['Distress_Label']].head())
```

3.5 Modelling

- **Logistic Regression**

```python
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score

# Defining features and target
X = data[features]
y = data['Distress_Label']

# Spliting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

# Creating an imputer object
imputer = SimpleImputer(strategy='median')

# Fiting the imputer on the training data and transform both training and testing data
X_train_imputed = imputer.fit_transform(X_train)
X_test_imputed = imputer.transform(X_test)

# Initializing and training Logistic Regression model
log_reg = LogisticRegression(max_iter=1000)

# Fit the model if the shapes match
log_reg.fit(X_train_imputed, y_train)

# Predicting and evaluating
y_pred_log_reg = log_reg.predict(X_test_imputed)
y_pred_proba_log_reg = log_reg.predict_proba(X_test_imputed)[:, 1]  # Probabilities for the positive class

print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_log_reg))
print("Logistic Regression Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_log_reg))

# Calculate AUC score
auc_log_reg = roc_auc_score(y_test, y_pred_proba_log_reg)
print(f"Logistic Regression AUC Score: {auc_log_reg}")
```

- Random Forest classifier

```
[30]
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Defining features (X) and target (y)
X = data[features]
y = data['Distress_Label']

# Imputing missing values using mean imputation
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Spliting the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

print(f"Training set size: {X_train.shape}")
print(f"Testing set size: {X_test.shape}")

# Initializing the Random Forest Classifier
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)

# Training the model
rf_classifier.fit(X_train, y_train)

# Making predictions
y_pred = rf_classifier.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print('Classification Report:')
print(classification_report(y_test, y_pred))

print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

- Gradient Boosting Machines (GBM)

```python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.impute import SimpleImputer

# Define features (X) and target (y)
X = data[features]
y = data['Distress_Label']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

print(f"Training set size: {X_train.shape}")
print(f"Testing set size: {X_test.shape}")

# Initialize the Gradient Boosting Classifier
gbm_classifier = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)

# Train the model
gbm_classifier.fit(X_train, y_train)

# Make predictions
y_pred = gbm_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print('Classification Report:')
print(classification_report(y_test, y_pred))

print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))

# Predicting probabilities for ROC AUC
y_pred_proba_gbm = gbm_classifier.predict_proba(X_test)[:, 1]

# Calculating ROC AUC score
from sklearn.metrics import roc_auc_score
auc_gbm = roc_auc_score(y_test, y_pred_proba_gbm)
print("Gradient Boosting AUC Score:", auc_gbm)
```

- Support Vector Machines (SVM)

```python
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.impute import SimpleImputer
from imblearn.over_sampling import SMOTE

# Define features (X) and target (y)
X = data[features]
y = data['Distress_Label']

# Handle missing values using mean imputation
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Generate synthetic samples using SMOTE
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)

# Split the resampled data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)

print(f"Training set size: {X_train.shape}")
print(f"Testing set size: {X_test.shape}")

# Initialize the Support Vector Classifier
svm_classifier = SVC(kernel='rbf', probability=True, random_state=42)

# Train the model
svm_classifier.fit(X_train, y_train)

# Make predictions
y_pred = svm_classifier.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

# Predicting probabilities for ROC AUC
y_pred_proba_svm = svm_classifier.predict_proba(X_test)[:, 1]

# Calculating ROC AUC score
auc_svm = roc_auc_score(y_test, y_pred_proba_svm)
print("SVM AUC Score:", auc_svm)

print('Classification Report:')
print(classification_report(y_test, y_pred))

print('Confusion Matrix:')
print(confusion_matrix(y_test, y_pred))
```

4 Evaluation

1. For Logistic regression Model: Precision: Ability of the classifier not to label a negative sample as positive.

- 0 (Non-Distressed): For non-distressed companies, the model correctly identifies them as non-distressed [precision value] of the time.
- 1 (Distressed): For distressed companies, the model correctly identifies them as distressed [precision value] of the time.

Recall: Ability of the classifier to find all the positive samples.

- 0 (Non-Distressed): The model correctly identifies [recall value] of all actual non-distressed companies.
- 1 (Distressed): The model correctly identifies [recall value] of all actual distressed companies.

F1-Score: Weighted harmonic mean of precision and recall. A good F1-score means a balance between precision and recall.

- Higher F1-scores are generally better, especially when there's an uneven class distribution.

Support: Number of samples of the true response that lie in that class.
Accuracy: Overall, the model correctly predicts the distress status of [accuracy value] of the companies in the test set.
Macro Avg: Average of precision, recall and F1-score between classes (gives equal weight to both classes).
Weighted Avg: Average of precision, recall and F1-score between classes (weighted by support, accounts for class imbalance).

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       233
           1       0.33      0.20      0.25         5

    accuracy                           0.97       238
   macro avg       0.66      0.60      0.62       238
weighted avg       0.97      0.97      0.97       238

Logistic Regression Confusion Matrix:
[[231   2]
 [  4   1]]
Logistic Regression AUC Score: 0.9682403433476395
```

2 Random Forest Classifier: Accuracy: The model correctly predicts the distress status of [accuracy value * 100]% of the companies in the test set. Precision: Ability of the classifier not to label a negative sample as positive.

- 0 (Non-Distressed): For non-distressed companies, the model correctly identifies them as non-distressed [precision value for class 0] of the time.
- 1 (Distressed): For distressed companies, the model correctly identifies them as distressed [precision value for class 1] of the time.

Recall: Ability of the classifier to find all the positive samples.

- 0 (Non-Distressed): The model correctly identifies [recall value for class 0] of all actual non-distressed companies.
- 1 (Distressed): The model correctly identifies [recall value for class 1] of all actual distressed companies.

F1-Score: Weighted harmonic mean of precision and recall. A good F1-score means a balance between precision and recall.

- Higher F1-scores are generally better, especially when there's an uneven class distribution.

Support: Number of samples of the true response that lie in that class.
Macro Avg: Average of precision, recall and F1-score between classes (gives equal weight to both classes).
Weighted Avg: Average of precision, recall and F1-score between classes (weighted by support, accounts for class imbalance).
Confusion Matrix:

- True Negative (Top Left): Number of non-distressed companies correctly predicted as non-distressed.

- False Positive (Top Right): Number of non-distressed companies incorrectly predicted as distressed.
- False Negative (Bottom Left): Number of distressed companies incorrectly predicted as non-distressed.
- True Positive (Bottom Right): Number of distressed companies correctly predicted as distressed.

```
Training set size: (554, 13)
Testing set size: (238, 13)
Accuracy: 0.96
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       228
           1       0.00      0.00      0.00        10

    accuracy                           0.96       238
   macro avg       0.48      0.50      0.49       238
weighted avg       0.92      0.96      0.94       238

Confusion Matrix:
[[228   0]
 [ 10   0]]
```

3 Gradient Boosting Machines (GBM): The model achieved high overall accuracy (96%), but the performance on the minority class (distressed) is poor.

- For class 0 (non-distressed), the model performs excellently with high precision, recall, and F1-score.
- For class 1 (distressed), the model has perfect precision (since all predicted class 1 instances are correct), but recall is very low (10%). It tells the model is not identifying many of the actual class 1 instances.
- The weighted average is skewed by the majority class due to class imbalance, showing good overall performance.

```
Training set size: (554, 13)
Testing set size: (238, 13)
Accuracy: 0.96
Classification Report:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       228
           1       1.00      0.10      0.18        10

    accuracy                           0.96       238
   macro avg       0.98      0.55      0.58       238
weighted avg       0.96      0.96      0.95       238

Confusion Matrix:
[[228   0]
 [  9   1]]
Gradient Boosting AUC Score: 0.9160087719298247
```

4 Support Vector Machines (SVM): Overall Performance: The SVM model performs well with an accuracy of 92%, showing correctness in predictions.

- Class 0 (non-distressed): The model has high precision (95%) but slightly lower recall (89%), meaning it is good at predicting class 0 correctly but misses some class 0 instances.
- Class 1 (distressed): The model has high recall (96%) and decent precision (90%), meaning it effectively identifies most of the class 1 instances but sometimes misclassifies some instances as class 1.

```
Training set size: (1085, 13)
Testing set size: (465, 13)
Accuracy: 0.92
SVM AUC Score: 0.9683855921223116
Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.89      0.92       227
           1       0.90      0.96      0.93       238

    accuracy                           0.92       465
   macro avg       0.93      0.92      0.92       465
weighted avg       0.93      0.92      0.92       465

Confusion Matrix:
[[202  25]
 [ 10 228]]
```